MATLAB QUICK REFERENCE

Arrays, matrices, plotting, file I/O, control flow

Basics

Command Window

```
x = 5;            % assign (semicolon suppresses output)
x = 5             % assign and display result
disp('Hello')     % print to console
clc               % clear command window
clear             % clear all variables
```

Help & Info

```
help sin          % quick help for function
doc sin           % open documentation
who               % list variables in workspace
whos              % list with details (size, type)
```

Operators

`+ - * / ^`	Arithmetic (matrix operations)
`.* ./ .^`	Element-wise operations
`== ~= < > <= >=`	Comparison operators
`&& || ~`	Logical AND, OR, NOT (scalars)
`& | ~`	Element-wise logical (arrays)

Variables & Types

Numeric Types

```
x = 3.14;         % double (default)
n = int32(42);    % 32-bit integer
z = 2 + 3i;       % complex number
tf = true;        % logical
```

Type Checking

`class(x)`	Return type name as string
`isa(x, 'double')`	Check if x is specific type
`isnumeric(x)`	True if numeric type
`ischar(x)`	True if character array
`islogical(x)`	True if logical type

Special Constants

`pi`	3.14159...
`Inf / -Inf`	Infinity
`NaN`	Not a Number
`eps`	Machine epsilon (~2.2e-16)
`i / j`	Imaginary unit

Arrays & Matrices

Creating Arrays

```
v = [1 2 3 4 5];          % row vector
v = [1; 2; 3];            % column vector
A = [1 2; 3 4];           % 2x2 matrix
r = 1:5;                  % [1 2 3 4 5]
r = 0:0.5:2;              % [0 0.5 1 1.5 2]
```

Built-in Constructors

```
zeros(3)          % 3x3 of zeros
ones(2, 4)        % 2x4 of ones
eye(3)            % 3x3 identity
rand(2, 3)        % 2x3 uniform random
linspace(0,1,5) % 5 evenly spaced [0..1]
```

Indexing & Slicing

```
A(2, 3)           % row 2, col 3
A(1, :)           % entire first row
A(:, 2)           % entire second column
A(1:2, 1:2)       % submatrix
A(end, :)         % last row
```

Matrix Operations

`A'`	Transpose (conjugate)
`A.'`	Transpose (no conjugate)
`inv(A)`	Matrix inverse
`det(A)`	Determinant
`eig(A)`	Eigenvalues and eigenvectors
`A \ b`	Solve Ax = b
`size(A)`	Dimensions [rows cols]
`numel(A)`	Total number of elements

Control Flow

if / elseif / else

```
if x > 0
    disp('positive')
elseif x == 0
    disp('zero')
else
    disp('negative')
end
```

for & while

```
for i = 1:10
    fprintf('i = %d\n', i);
end
while x > 0
    x = x - 1;
end
```

switch

```
switch grade
    case 'A'
        disp('Excellent')
    case {'B', 'C'}
        disp('Good')
    otherwise
        disp('Try harder')
end
```

Loop Control

`break`	Exit the innermost loop
`continue`	Skip to next iteration
`return`	Exit function immediately

Functions

Function File

```
% Save as myfunc.m
function result = myfunc(x, y)
    result = x.^2 + y.^2;
end
```

Multiple Outputs

```
function [mn, mx] = minmax(v)
    mn = min(v);
    mx = max(v);
end
[lo, hi] = minmax([3 1 4 1 5]);
```

Anonymous Functions

```
f = @(x) x.^2 + 1;
f(3)                      % returns 10
g = @(x,y) x + y;
arrayfun(f, [1 2 3])      % apply to each element
```

Useful Built-ins

`sum(v)`	Sum of elements
`mean(v)`	Mean value
`max(v) / min(v)`	Maximum / minimum
`sort(v)`	Sort ascending
`find(v > 3)`	Indices where condition is true
`length(v)`	Length of vector

Plotting

2D Plots

```
x = 0:0.1:2*pi;
plot(x, sin(x), 'r-', 'LineWidth', 2)
xlabel('x'); ylabel('sin(x)')
title('Sine Wave'); grid on
legend('sin(x)')
```

Multiple Plots

```
hold on
plot(x, sin(x), 'b-')
plot(x, cos(x), 'r--')
hold off
subplot(1,2,1); plot(x, sin(x))
subplot(1,2,2); plot(x, cos(x))
```

Other Plot Types

`bar(x, y)`	Bar chart
`histogram(data)`	Histogram
`scatter(x, y)`	Scatter plot
`pie(data)`	Pie chart
`surf(X, Y, Z)`	3D surface plot
`imagesc(A)`	Display matrix as image

Save Figure

```
saveas(gcf, 'plot.png')
exportgraphics(gcf, 'plot.pdf')
```

File I/O

Text Files

```
data = readmatrix('data.csv');
writematrix(A, 'output.csv')
T = readtable('data.csv');
writetable(T, 'output.csv')
```

MAT Files

```
save('workspace.mat')           % save all variables
save('data.mat', 'x', 'y')      % save specific vars
load('data.mat')                % load into workspace
S = load('data.mat');           % load into struct
```

Low-Level File I/O

```
fid = fopen('log.txt', 'w');
fprintf(fid, 'Value: %f\n', 3.14);
fclose(fid);
lines = readlines('log.txt');
```

String Operations

String vs Char Array

```
s = "Hello";        % string (double quotes)
c = 'Hello';        % char array (single quotes)
s + " World"        % "Hello World" (string)
[c, ' World']       % 'Hello World' (char concat)
```

String Functions

`strlength(s)`	Length of string
`upper(s) / lower(s)`	Case conversion
`contains(s, pat)`	True if pattern found
`replace(s, old, new)`	Replace substring
`split(s, delim)`	Split into array
`join(arr, delim)`	Join string array
`strip(s)`	Remove leading/trailing whitespace

Formatting

```
sprintf('x = %.2f', 3.14159)   % "x = 3.14"
fprintf('i = %d\n', 42)         % print to console
num2str(3.14)                   % number to string
str2double("3.14")              % string to number
```

Cell & Struct

Cell Arrays

```
C = {1, 'hello', [1 2 3]};  % mixed types
C{2}                         % access: 'hello'
C{end+1} = true;             % append element
cellfun(@length, C)          % apply func to each
```

Structs

```
s.name = 'Alice';
s.age = 30;
s.scores = [90 85 92];
fieldnames(s)          % {'name','age','scores'}
rmfield(s, 'age')      % remove field
```

Struct Arrays

```
people(1).name = 'Alice'; people(1).age = 30;
people(2).name = 'Bob';   people(2).age = 25;
{people.name}             % {'Alice', 'Bob'}
[people.age]              % [30, 25]
```

Common Patterns

Vectorized Operations

```
% Avoid loops — use vectorization
v = 1:1000;
result = sum(v.^2);          % fast
idx = v(v > 500 & v < 600);  % logical indexing
```

Table Operations

```
T = table([25;30], ["A";"B"], 'VariableNames', ...
          {'Age','Grade'});
T.Age          % access column
T(T.Age > 25, :)  % filter rows
```

Error Handling

```
try
    result = riskyFunction(x);
catch ME
    fprintf('Error: %s\n', ME.message);
end
```

Timing Code

```
tic
heavyComputation();
toc              % prints elapsed time
```